Journal of Computer and System Sciences 79 (2013) 111–121

Contents lists available at SciVerse ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Space and speed tradeoffs in TCAM hierarchical packet classification ☆

Alexander Kesselman [a], Kirill Kogan [b], Sergey Nemzer [c], Michael Segal [d,*]

[a] Google Inc., United States
[b] Computer Science Department, University of Waterloo, Canada
[c] Compugen, Israel
[d] Department of Communication Systems Engineering, Ben-Gurion University, Israel

ARTICLE INFO

Article history:
Received 19 August 2010
Received in revised form 21 May 2012
Accepted 4 June 2012
Available online 13 June 2012

Keywords:
TCAM hierarchical packet classification
Dynamic programming
Lookups and space tradeoff

ABSTRACT

Traffic classification in the Internet is a crucial mechanism necessary to support network services. Using Ternary Content-Addressable Memories (TCAMs) to perform high-speed packet classification has become the de facto standard in industry. TCAMs concurrently match the packet headers against the rules in a classification database providing high throughput unparalleled by software-based solutions. The complexity of packet classification policies has been growing rapidly as the number of Internet services continues to increase. Many complex classification policies are naturally represented in a hierarchical fashion, where different layers perform classification based on the administrative domain and the traffic QoS parameters. However, multiple levels of classification hierarchy incur high lookup latency while high TCAM memory requirements of flattened classification policies is a major issue since TCAMs have very limited capacity. In this paper we focus on the fundamental tradeoff between the TCAM space and the number of lookups in the TCAM classification policies. We consider two optimization problems of dual nature: the first problem is to minimize the number of TCAM entries subject to the constraint on the maximum number of levels in the policy hierarchy; the second problem is to minimize the number of levels in the policy hierarchy subject to the constraint on the maximum number of TCAM entries. We propose efficient algorithms for these problems, which do not require any hardware changes. To the best of our knowledge, this is the first work to study these problems. We also show experimental results that support our findings.

1. Introduction

Packet classification is a crucial function for a variety of emerging network services such as Quality of Service (QoS), firewalling, and traffic monitoring and accounting, to name just a few.

An important characteristic influencing the network services is so-called network hierarchy, that is the degree of concentration of traffic flows at interconnection points within the network. Hierarchies are important since they help to determine the sizes of networks, including the routing and addressing configurations as well as the scaling of network technologies, performance, and service levels. Furthermore, service-level agreements can include support for delay-constrained applications and may contain a variety of capacity and control mechanisms, such as traffic shaping and policing at multiple levels in the network.

☆ The preliminary version of this paper has appeared in IEEE Sarnoff Symposium, 2008. The work on this paper has been partially supported by US Air Force European Office of Aerospace Research and Development, grant FA8655-09-1-3016, Deutsche Telecom, European project FLAVIA and Israeli Ministry of Industry, Trade and Labor (consortium CORNET).

* Corresponding author.
E-mail addresses: alx@google.com (A. Kesselman), kkogan@uwaterloo.ca (K. Kogan), sergeyn@compugen.co.il (S. Nemzer), segal@cse.bgu.ac.il (M. Segal).

0022-0000/$ – see front matter
http://dx.doi.org/10.1016/j.jcss.2012.06.001

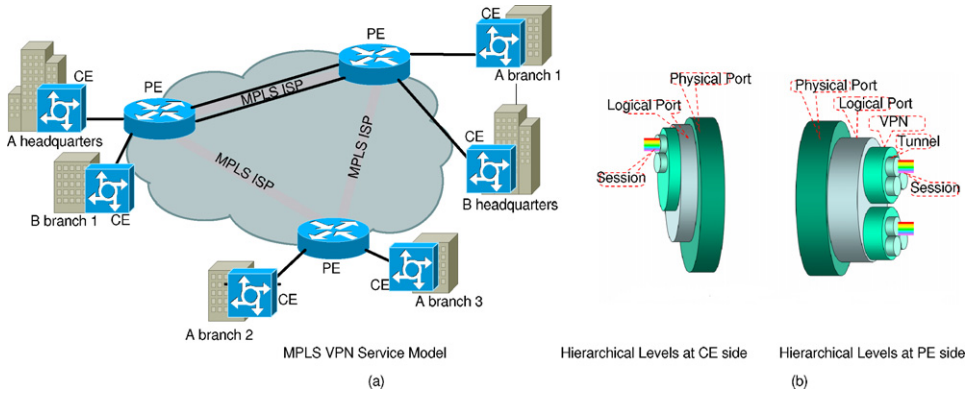

Fig. 1. Application of hierarchical QoS for MPLS VPN service [30].

Network devices maintain service policies under which incoming or outgoing packets are classified by matching against a hierarchical set of rules. For example, bandwidth can be shared between different users at the top level matching traffic according to the administrative affiliation while the second tier of the hierarchy can further classify the traffic according to the QoS class based on the Differentiated Services Code Point (DSCP), protocol type, or some other criteria. In addition, each rule can also specify a set of actions to be taken on packets matching this rule.

Several types of services such as *Multi Play*, *MPLS VPN*, *VPDN*, to name just a few, require implementation of hierarchical policies [30]. For instance, consider the following implementation of MPLS VPN service that is described in [30]. The *Customer Edge* (CE) side of the *Provider Edge* (PE) router links the branches of an enterprise, where each branch has a special bandwidth requirement. The enterprise has a limit on the outgoing traffic and the tunnels between PE routers bear traffic from different enterprises, which have different QoS requirements. The tunnels also have bandwidth limitations of their own. Fig. 1(a) depicts the MPLS VPN service model. Though for the CE side the traditional QoS can improve the service quality of each branch, it cannot manage the traffic of branches and that of the whole enterprise on PE. The problem is that for tunnels between PEs, the traditional QoS applies to either the tunnel or the internal traffic separately without being able relate between the two. On the other hand, the hierarchical QoS model can meet the QoS requirement of MPLS VPN service. Fig. 1(b) demonstrates a hierarchical structure of QoS policy that can be applied on CE and PE sides. Observe that on a PE side the hierarchy has five levels in depth. Therefore, supporting hierarchical service policies is a challenging task, which requires to perform hierarchical matching at the line rate.

Hierarchical classification polices are widely used in today's commercial routers [27–30]. However, most hierarchies have only a few levels due to the speed (line rate) constraints. In this work we study how to overcome this performance limitation. Specifically, we explore several interesting tradeoffs between the speed and the required space for wire-speed hierarchical packet classifiers that are implemented in *Ternary Content-Addressable Memory* (TCAM).

A TCAM is a memory device that stores data as a massive array of fixed-width ternary entries. A ternary entry is a string of bits where each bit is either 0, 1 or $*$ ("don't care"). The TCAM searches the packet in parallel against all the ternary entries stored in the memory and produces the first rule that matches the packet. Remarkably, TCAM guarantees that each lookup is done in constant time. Usually each TCAM entry is wide enough to contain the concatenation of all the packet fields to be matched, possibly having room for some extra bits. If a matching rule consists solely of fields that specify exact or prefix matches, then it can be represented by a TCAM entry in a straightforward manner (a prefix match field is padded with the appropriate number of $*$'s in the least significant bits). A range value may be converted to multiple prefixes or exact entries to fit the TCAM format.

2. Our results

We study two fundamental problems dealing with hierarchical packet classification using TCAM. We are given a policy P with d levels of hierarchy and the goal is to convert P to an equivalent policy P' that needs to fulfill certain constraints minimizing the number of lookups or TCAM entries required. Our algorithms do not require any hardware modifications being very easy to deploy.

In the space optimization problem, we aim to minimize the number of TCAM entries required by P' subject to the limit on the maximum number of hierarchy levels. The motivation behind this problem is to reduce the required TCAM space allowing packet classification in wire-speed without incurring a possibly huge memory blowup if P is flattened to a single level. We propose two dynamic programming algorithms for this problem: the first algorithm is more efficient but is restricted only to policies that match disjoint fields in a packet header at different levels of the hierarchy; the second algorithm is slightly more complicated and can process general policies.

In the speed optimization problem, our goal is to minimize the number of hierarchy levels in P' subject to the limit on the maximum number of TCAM entries. The rationale behind this problem is to utilize the available TCAM capacity as

efficiently as possible to reduce the number of lookups. We propose an algorithm that applies one of our speed optimization algorithms as a subroutine. This algorithm adds a factor of $O(\log d)$ to the running time of the corresponding speed optimization algorithm. Please observe that the suggested algorithms are orthogonal to different approaches that minimize TCAM entries such as removing of redundant rules, conversion of ranges to prefixes, etc. (see Section 3) at a single hierarchical level. All our algorithms operate on already optimized classification rules at each hierarchical level and transparent to any specific TCAM HW architecture.

3. Related work

Designing algorithms that scale to millions of rules and millions of searches per second has been and continues to be an important line of research. Many software-based sophisticated approaches have been proposed in the past few years including Recursive Flow Classification [9], Crossproducting [6,21,23], HyperCuts [20], Extended Grid-of-Tries [2] and Aggregated Bit Vector [3], to name just a few. Comprehensive surveys on this subject can be found in [7,11,22,24]. The complexity bounds derived by means of computational geometry imply that any software-based packet classifier with N rules and $k > 2$ fields, uses either $O(N^k)$ space and $O(\log N)$ time or $O(N)$ space and $O(\log^{k-1} N)$ time [18]. Thus, many software-based approaches are either too slow or too memory intensive for $k > 2$. Though packet classification algorithms using decision trees achieve better time–space tradeoffs (see [10,25]), they exploit statistical characteristics that are not reliable in general.

Due to the inherent limitations of software-based approaches, industry has increasingly employed hardware-based *Ternary Content-Addressable Memory* (TCAM) for performing packet classification making it the dominant method [31–33]. A large class of packet classification systems that require up to a few hundred thousand rules has adopted TCAM for packet classification at multigigabit speeds [4,8].

Some previous works consider TCAM space minimization for a packet classifier, for example [13,17]. For the best of our knowledge all of them deal with a single level of hierarchy concentrating on elimination of overlapping rules and representation of filters ranges. Several schemes for converting ranges to TCAM rules have been proposed in [5,12–17]. Reducing of TCAM power consumption and increasing of throughput is explored in [1,19,26].

4. Paper organization

The rest of the paper is organized as follows. The model description appears in Section 5. The algorithms for space and speed optimization are presented in Sections 6 and 8, respectively. The generalization of space optimization problem is considered in Section 7. Finally, we conclude with Section 10.

5. Model description

In this section we introduce the formal notation and define the hierarchical speed and space optimization problems.

5.1. Notation

A packet header contains k fields, where a field H_i $(1 \leqslant i \leqslant k)$ is a string of W_i bits. In an IPv4 packet, classifiers usually check the following six fields: the Type of Service (8 bits), the Destination Address (32 bits), the Source Address (32 bits), the Destination Port (16 bits), the Source Port (16 bits), and the Protocol Type (8 bits). Note that classifiers may access other fields besides TCP/IP header such as MAC or application headers. Packets are matched according to classification rules stored in a classification database.

The classification database of a router consists of a finite set of n rules, $R_1 \ldots R_n$. Each rule R specifies matchings for one or more (up to k) fields. For each header field H_i, a rule can specify a filter F_i of length $|F_i|$ $(|F_i| \leqslant |H_i|)$, which can be any of two kinds of matches: exact match or prefix match.

1. A packet header field H_i exactly matches the filter F_i if and only if $H_i = F_i$.
2. A packet header field H_i is a prefix match for the filter F_i if and only if the $|F_i|$ bits of H_i are equal to F_i.

A packet p matches rule R if each of p's header fields matches the corresponding filter of R if any. The header fields for which filters are not specified by the rule are matched in TCAM by a wildcard filter ("don't care"). Since a packet may match multiple rules, the classification problem is to determine the first matching rule in an ordered sequence of rules.

There is also a third type of matching, so-called range matching, where the header value should fall into a contiguous interval specified by the filter. In typical packet classifiers, such fields as the source and destination port numbers are represented as ranges rather than prefixes. Though range rules cannot be directly stored in TCAMs, they are usually converted to a corresponding set of prefixes each of which is stored in a separate TCAM entry. In this paper we deal with classification rules that reside in a TCAM device and assume that all filters are exact or prefix match (the classification database may undergo a conversion if necessary [5,12–17]). Such transformation of rules is orthogonal for our future discussion since our algorithms manipulate with already converted rules on each hierarchical level.

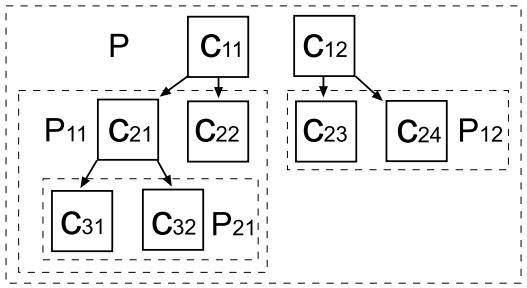

Fig. 2. An example of policy and its graph G_p.

We define a *class* C to be an ordered set of rules and a set of *actions* to be taken on the packet. For instance, a QoS action may be packet marking with a pre-defined DSCP value while a security action may be packet accept or reject. We denote by $|C|$ the number of rules or *cardinality* of C. We say that the class is matched if the first matching rule belongs to this class. A *policy* P is an ordered set of classes. The last class of a policy is usually so-called *default* class matching all packets that have not been matched by the other classes. Note that a global order of the rules is obtained by listing the rules in the corresponding classes.

The action of a class can also apply another policy in recursive manner, creating a *hierarchical policy*. Each recursive application creates a new *level of hierarchy*. A hierarchical policy can be viewed as a directed and acyclic graph G_P with classes acting as nodes and each edge representing a recursive policy application (see Fig. 2). A directed path $\mathcal{C} = C_1 \to C_2 \to \cdots \to C_d$ from a *root* class with no incoming edges to a *terminal* class with no outgoing edges is called a *class chain*. The length of the longest class chain is defined as the *policy depth*. In order to match all rules on a class chain $\mathcal{C}$, a packet has to match rules $R_1 \in C_1$, $R_2 \in C_2$, …, $R_d \in C_d$. Observe that the total number of hierarchical class chains is equal to the number of terminal classes.

In the actual implementation of a hierarchical classifier, a special header field H_0 is added to a packet at all levels of the hierarchy to identify the class sub-chain $\mathcal{S} = C_1 \to C_2 \to \cdots \to C_l$ $(1 < l \leqslant d)$ matched by the packet up to and including level l. In this way, the classifier can track the path of the packet in G_P and take the appropriate actions as matching proceeds. Furthermore, all the rules corresponding to the children of the class C_l in G_P have a common filter F_0 that identifies the class sub-chain $\mathcal{S}$. As a result, different class chains in a hierarchical classifier are independent in the sense that distinct instances of the same class are created for each node in G_P. Therefore, in order to minimize the number of entries across different hierarchical chains it is sufficient to consider separately each one of the chains. We assume that the size of F_0 filter is at least as the cardinality of all hierarchical class chains in the policy graph. A value of H_0 at each hierarchical level (from the second) is identified as result of the previous TCAM lookup. At the first level the value of H_0 is known a priori and its value is identical for all hierarchical chains of the same policy.

Hierarchical policies allow a high degree of flexibility and modularity in policy definition. However, a separate TCAM lookup needs to be performed for each level of the hierarchy in the process of classification, which may incur large delays for policies with high depth. To speed up the classification process, a hierarchical policy can be converted to an equivalent policy of lower depth. At the same time, such a conversion can significantly increase the number of TCAM entries required to store the merged rules. We define the *TCAM space* M of a policy as the total number of rules in all levels of the classification hierarchy.

The operations defined below deal with policy flattening. We will describe how to *intersect* two filters F_i and F'_i specified for a common header field H_i. Suppose without loss of generality that $|F_i| \leqslant |F'_i|$. If F_i is a prefix of F'_i, then a resulted filter $F_i \cap F'_i$ will consist of filter F'_i. Otherwise, a resulted filter $F_i \cap F'_i$ is an *empty filter* that does not match any value of header field H_i. The merge of rules $R \otimes R'$ is defined as a set of intersections of the corresponding filters in R and R'. If at least one empty filter is produced during this operation then the merged rule $R \otimes R'$ is called an *empty rule* that does not match any packet. We define the *merge* of two classes $C \otimes C'$ as a class consisting of all possible merges of rules $R \otimes R'$ that are not empty, where $R \in C$ and $R' \in C'$ that sequentially applies actions of C and C'. For a class chain $\mathcal{C} = C_1 \to C_2 \to \cdots \to C_d$ we define a *virtual class* $C_{1,d}$ as $C_1 \otimes C_2 \otimes \cdots \otimes C_d$ that represents the merge of all classes in $\mathcal{C}$.

Observe that $|C_1 \otimes C_2 \ldots C_n| \leqslant |C_1||C_2| \ldots |C_n|$. Let C_1 consist of three rules $[ToS = 1]$, $[ToS = 2]$ and $[ToS = 3]$ and C_2 consist of two rules $[DstPort = 21]$ and $[DstPort = 80]$. In this case, $|C_1 \otimes C_2| = 6$ as the filters are specified for disjoint header fields and $C_1 \otimes C_2$ contains the following rules: $[ToS = 1, DstPort = 21]$, $[ToS = 2, DstPort = 21]$, $[ToS = 3, DstPort = 21]$, $[ToS = 1, DstPort = 80]$, $[ToS = 2, DstPort = 80]$, $[ToS = 3, DstPort = 80]$. At the same time, the TCAM space required to represent the merge of classes may be smaller than the cardinality of the classes themselves if filters are specified for common header fields. For instance, suppose that C_1 contains three rules $[ToS = 1]$, $[ToS = 2]$ and $[ToS = 3]$ and C_2 contains two rules $[ToS = 3]$ and $[ToS = 4]$. We obtain that $|C_1 \otimes C_2| = 1$ and $C_1 \otimes C_2$ includes just one rule $[ToS = 3]$.

5.2. Problem statement

We say that two packet classifiers are (semantically) equivalent if and only if they apply the same actions on each packet. Next, we define the optimization problems studied in this paper.

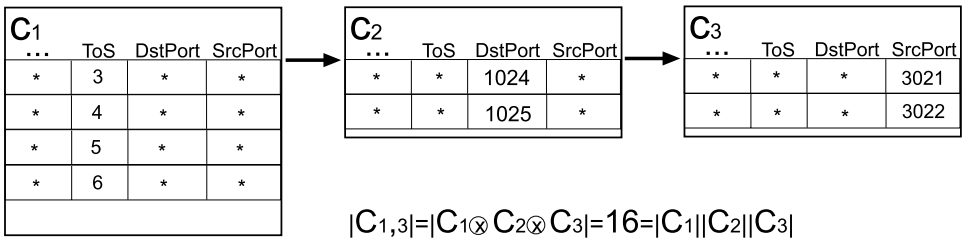

Fig. 3. Example of well-structured hierarchy.

Input: *policy P of depth d, integer l ($l < d$)*
Output: *policy P' equivalent to P of depth l*

- **Step 1: Merging long chains.** For each class chain $\mathcal{C} = C_1 \to C_2 \to \cdots \to C_d$ in P such that $d > l$ create a virtual class $C_{1,d} = C_1 \otimes C_2 \otimes \cdots \otimes C_d$ that represents the merge of all classes in $\mathcal{C}$ (without merging the rules).
- **Step 2: Splitting merged chains.** Split each merged virtual class $l-1$ times using the level splitting algorithm (see Fig. 6).
- **Step 3: Merging rules.** For each virtual class, merge the rules of the corresponding merged classes.
- **Step 4: Creating output policy.** Output P' as a union of all the original chains of length at most l and the converted long chains.

Fig. 4. Space optimization algorithm (*SOAW*) for well-structured hierarchies.

Hierarchical space optimization problem: Given a hierarchical policy P with depth $d > 1$, the goal is to convert P to an equivalent policy P' with depth of at most l ($l < d$) that minimizes the required TCAM space.

Hierarchical speed optimization problem: Given a hierarchical policy P that requires TCAM space of M, the aim is to convert P to an equivalent policy P' that minimizes the policy depth subject to the constraint that the TCAM space cannot exceed the available TCAM space A ($A > M$).

For now we assume that a set of classified packet headers is known a priori.

6. Space optimization

In this section we consider the case where the values of classified packet headers are known a priori and are not changed during classification process. We consider the problem of minimizing the TCAM space subject to the constraint on the policy depth. First, we present an algorithm for the case where rules in any class chain apply only to disjoint header fields. Then we propose an algorithm for the general case where we impose no restrictions on the policy whatsoever.

6.1. Well-structured hierarchies

In this section we study *well-structured* hierarchical policies in which rules in any class chain apply only to disjoint header fields. The example of well-structured hierarchy is presented on Fig. 3. We can learn from the same figure the importance of decision which sub-chains should be merged. At one hand, if we merge the classes C_2 and C_3 then the cardinality of flattened hierarchical chain is $|C_1| + |C_2 \otimes C_3| = 4 + 4 = 8$. From the other hand, if we merge the classes C_1 and C_2 then the cardinality of the flattened chain is $|C_1 \otimes C_2| + |C_3| = 8 + 2 = 10$.

Well-structured hierarchies have the following important property, which allows us to use a fast dynamic programming algorithm operating merely with cardinalities of merged classes without actually merging the rules themselves until the final stage.

Observation 1. *In a well-structured hierarchy, for any class chain $\mathcal{C} = C_1 \to C_2 \to \cdots \to C_d$ the cardinality of the classes' merge in $\mathcal{C}$, that is $|C_1 \otimes C_2 \otimes \cdots \otimes C_d|$, equals the product of the cardinalities of the individual classes $|C_1||C_2| \ldots |C_d|$.*

The space optimization algorithm for well-structured hierarchies (*SOAW*) is presented on Fig. 4. *SOAW* proceeds by first merging all chains of length greater than l into a single *virtual* class without actually merging the rules. Then each virtual merged class is split $l-1$ times in an optimal way using a level splitting algorithm based on the dynamic programming technique. Finally, for each virtual class the rules of the corresponding merged classes are merged to produce the output policy. Observe that class chains of length smaller than l are left untouched by *SOAW*. The running time of *SOAW* is output sensitive. Basically, it builds the optimized policy according to the output of level splitting algorithm, which is applied to all long chains in P. Recall that the total number of class chains equals the number of terminal classes.

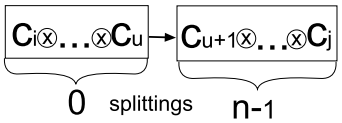

Fig. 5. Dynamic programming for well-structured policy.

Input: *hierarchical chain $\mathcal{C}$ consisting of d classes, integer l ($l < d$)*
Output: *chain $\mathcal{C}'$ of length l*

- **Step 1: Initialization.** Initialize $V(i, j, n)$ for $n = 0$ and $n > j - i$ according to Eq. (1).
- **Step 2: Calculation.** Calculate all values $V(i, j, n)$ starting from $n = 1$ up to $n = l - 1$ for $n \leqslant j - i$ using the recurrence Eq. (2) and record the splitting of minimum cost.
- **Step 3: Reconstructing the optimal solution.** Construct the chain $\mathcal{C}'$ by splitting the classes of C with respect to the optimal solution of minimum cost $V(1, d, l - 1)$.

Fig. 6. Level splitting algorithm (*LSA*) for well-structured hierarchies.

Now we describe how to divide the level splitting problem into two sub-problems and combine solutions to these sub-problems into a solution to the original problem. For a merged class $C_{1,d} = C_1 \otimes C_2 \otimes \cdots \otimes C_d$, we denote by $V(i, j, n)$ $(1 \leqslant i < d,\ i < j \leqslant d)$ the cost of an optimal solution for the problem of n splittings in the merged sub-class $C_{i,j} = C_i \otimes C_{i+1} \otimes \cdots \otimes C_j$. The cost is measured as the cumulative cardinality of the resulting $n + 1$ sub-classes when $n \leqslant j - i$.

We define initial values for the special case of $n = 0$ where nothing needs to be done and the special case of $n > j - i$ where no feasible solution exists:

$$V(i, j, n) = \begin{cases} |C_{i,j}| : n = 0, \\ \infty : n > j - i. \end{cases} \tag{1}$$

The main recurrence relation is defined as follows for $n > 0$ and $n \leqslant j - i$:

$$V(i, j, n) = \min_u \big(V(i, u, 0) + V(u + 1, j, n - 1)\big) \tag{2}$$

for $i \leqslant u \leqslant j - n$.

Basically, in order to make n level splittings we consider all possibilities for the first splitting and perform exhaustive search over all the remained at most $n - 1$ splittings in $C_{u+1,j}$ sub-class. Fig. 5 demonstrates this process.

Our aim is to minimize the overall cost of the produced solution. The level splitting algorithm (*LSA*) appears on Fig. 6.

It is easy to see that the space complexity of *LSA* is $O(d^2 l)$ and the running time is $O(\max(dl^2, d^2 l))$, which is very reasonable since d and l are typically small numbers. The next theorem shows the correctness of *LSA*.

Theorem 6.1. *The level splitting algorithm* (*LSA*) *finds an optimal solution of minimum cost for the problem of n splittings* ($n \geqslant 0$) *in a class formed as the merge of d classes* ($d > n$).

Proof. The proof is by induction on the number of splittings n. Clearly, *LSA* finds an optimal solution for $n = 0$ since it just returns the original input as no splittings are necessary.

Assume that *LSA* finds an optimal solution for the number of splittings of at most n and let us show that it also finds an optimal solution for $n + 1$ splittings. Note that *LSA* considers all options for making the first splitting, one of which necessarily corresponds to an optimal solution.

Having done an optimal first splitting, it must be the case that the rest of this optimal solution consists of an independent optimal solution for the right sub-class with at most n splittings, see Fig. 5. By the induction hypothesis, *LSA* finds an optimal solution for the right sub-class with at most n splittings. Therefore, *LSA* finds an optimal solution for at most $n + 1$ splittings. □

6.2. Arbitrary hierarchies

In this section we deal with the general case of arbitrary hierarchies. Unfortunately, such hierarchies require to calculate the actual merge of the rules for the classes' merge in order to obtain its cardinality as demonstrated by the next observation.

Observation 2. *In an arbitrary hierarchy, for any class chain $\mathcal{C} = C_1 \to C_2 \to \cdots \to C_d$ we have that the cardinality of the merge of all classes in $\mathcal{C}$, that is $|C_1 \otimes C_2 \otimes \cdots \otimes C_d|$, is bounded from above but may not be equal to the product of the cardinalities of the individual classes $|C_1||C_2| \ldots |C_d|$.*

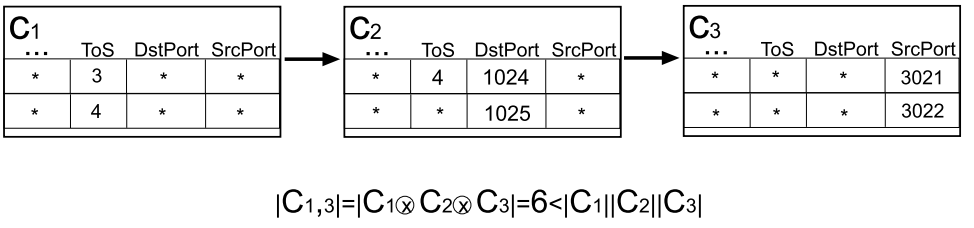

Fig. 7. Example of arbitrary hierarchy.

Input: *policy* P *of depth* d, *integer* l ($l < d$)
Output: *policy* P' *equivalent to* P *of depth* l

- **Step 1: Merging long chains.** Merge all chains of length d' greater than l using the level merging algorithm with the number of merges $m = d' - l$.
- **Step 2: Creating output policy.** Output P' as a union of all the original chains of length at most l and the converted long chains.

Fig. 8. Space optimization algorithm (*SOAG*) for general hierarchies.

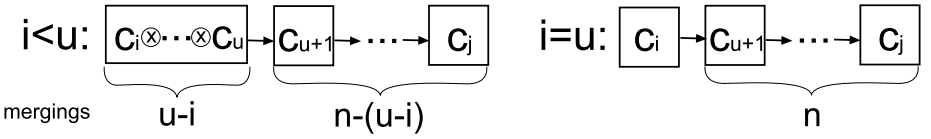

Fig. 9. Dynamic programming for a chain with arbitrary structure.

The example of arbitrary hierarchy is shown on Fig. 7.

That necessitates a slightly more complicated dynamic programming algorithm for level merging compared to the level splitting algorithm used for well-structured hierarchies. The space optimization algorithm for general hierarchies (*SOAG*) appears on Fig. 8. *SOAG* merges all chains of length greater than l by running a level merging algorithm based on the dynamic programming technique. The main component of *SOAG* is the level merging algorithm, which processes all long chains in P.

In what follows we present a way of dividing the level merging problem into two sub-problems and combining solutions to these sub-problems into a solution to the original problem. For a class chain $\mathcal{C} = C_1 \to C_2 \to \cdots \to C_d$, define $V(i, j, n)$ ($1 \leqslant i < d$, $i < j \leqslant d$) as the cost of an optimal solution for the problem of merging n levels in the class sub-chain $\mathcal{C}_{i,j} = C_i \to C_{i+1} \to \cdots \to C_j$. We estimate the cost of a solution as the total cardinality of the produced $j - i - n + 1$ sub-classes.

Initial values are set for the following cases: $n = 0$, where no merges have to be performed, $n > j - i$, which permits no feasible solution, and $n = j - i$, where all levels are completely merged. Note that there is no need to merge more than $d - l$ levels to obtain the desired solution.

$$V(i, j, n) = \begin{cases} \sum_{u=i}^{j} |C_u| : n = 0, \\ \infty : n > j - i, \\ |C_{i,j}| = |C_i \otimes \cdots \otimes C_j| : n = j - i,\ n \leqslant d - l. \end{cases} \tag{3}$$

We specify the main recurrence relation for $n > 0$ and $j - i > n$ in the following way:

$$V(i, j, n) = \min_{u} \big(V(i, u, u - i) + V\big(u + 1, j, n - (u - i)\big) \big) \tag{4}$$

for $i \leqslant u \leqslant i + n$.

Essentially, we cover all options for a leftmost merged class sub-chain $\mathcal{C}_{i,u}$ with $u - i$ mergings. Then we consider all possible partitions of the remaining $n - (u - i)$ mergings of the class sub-chain $\mathcal{C}_{u+1,j}$. The goal is to minimize the total cost of the resulting solution. The level merging algorithm (*LMA*) can be found in Fig. 10. We obtain that the space complexity of *LMA* is $O(md^2)$ and the running time is $O(\max(md^2, m^2 d))$. The subsequent theorem demonstrates the correctness of *LMA*.

Theorem 6.2. *The level merging algorithm* (*LMA*) *finds an optimal solution of minimum cost for the problem of merging* ($n \geqslant 0$) *levels in a chain of length* d ($d > n$).

Proof. The proof is by induction on the length of sub-chain d and number of mergings n. Consider the case when $d = 1$. Obviously, *LMA* finds an optimal solution for $n = 0$ since it just returns the original input as no mergings need to be done. Since $d > n$, the induction base for the case $d = 1$ follows.

Input: *chain* $\mathcal{C}$ (*of length* d), *integer* m ($m < d$)
Output: *chain* $\mathcal{C}'$ *of length* $d - m$

- **Step 1: Initialization.** Initialize $V(i, j, n)$ for $n = 0$, $n > j - i$, and $n = j - i$ ($n \leqslant m$) according to Eq. (3).
- **Step 2: Calculation.** Calculate all values $V(i, j, n)$ starting from $n = 1$ up to $n = m$ for $n > j - i$ using the recurrence Eq. (4) and record the merging of minimum cost at each stage.
- **Step 3: Reconstructing the optimal solution.** Construct the chain $\mathcal{C}'$ by merging the classes of $\mathcal{C}'$ with respect to the optimal solution of minimum cost $V(1, d, m)$.

Fig. 10. Level merging algorithm (*LMA*) for general hierarchies.

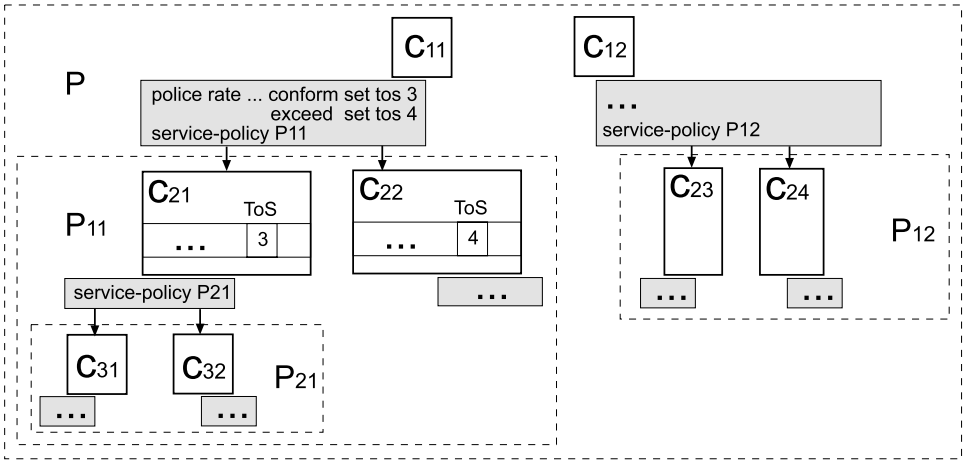

Fig. 11. Motivation for generalized space optimization problem.

Assume the theorem holds for the length of sub-chain less than d and let us prove it for the case when the length of sub-chain equals d. Obviously, *LMA* finds an optimal solution for $n = 0$ since it just returns the original input as no mergings need to be done. Suppose that *LMA* finds an optimal solution for the number of mergings of at most n and let us prove that it finds an optimal solution for $n + 1$ mergings as well. Observe that *LMA* examines all possibilities for making z ($1 \leqslant z \leqslant n + 1$) mergings in the leftmost class sub-chain of length less than d. It must be the case that one of these leftmost merged class sub-chains is a part of an optimal solution. Having merged the leftmost class sub-chain of length less than d (that is a part of optimal solution), the other at most $n + 1 - z$ mergings in this solution are independent for the right class sub-chain of length less than d. According to the induction hypothesis, *LMA* finds an optimal solution in the right class sub-chain of length less than d with at most $n + 1 - z \leqslant n$ mergings if $z > 0$. If the first merging is an empty merging (i.e. $z = 0$), then the above argument is applied to the right class sub-chain of length less than d, see the right part of Fig. 9. Hence, *LMA* finds an optimal solution for at most $n + 1$ mergings. □

7. Generalization of space optimization problem

In this section we discuss a generalization of the space optimization problem in which we obtain an additional constraint that the inner protocol header fields cannot be extracted until the outer protocol header fields have been parsed. The rationale behind this problem is that certain levels of the hierarchy cannot be "merged" during optimization of a hierarchical chain as classifying complex protocols based on the analysis of encapsulated content often requires sequential parsing of nested protocol headers. For instance, the application header cannot be extracted directly if the TCP header contains variable length options. Moreover, packet header fields can be dynamically changed during classification process as a result of class actions.

A *barrier* is the level of hierarchical chain where two neighboring classes belonging to it could not be "merged". The classifier has to issue a separate TCAM lookup for each barrier in hierarchical chain. For instance, on Fig. 11 we consider a packet that is matched to the class C_{11}. For correctness of classification process we have to update TOS value to 3 or 4 according to police action of C_{11} that defines a barrier in all hierarchical chains initiated from C_{11}. That means C_{11} could not be merged with C_{21} during optimization of hierarchical chain $\mathcal{C} = C_{11} \to \cdots \to C_{31}$.

We will show that the generalized version of space optimization algorithms is simply extendable. The definition of *virtual sub-class* $C_{i,j}$ previously being defined as $C_i \otimes C_{i+1} \otimes \cdots \otimes C_j$ now should take into account permanently located barriers. More precisely, if there is a barrier at the k-th position in a hierarchical sub-chain $\mathcal{C}_{i,j}$, $i \leqslant k < j$ then the virtual sub-class $C_{i,j}$ is defined as $C_{i,k} \to C_{k+1,j}$ and its cardinality $|C_{i,j}|$ is equal to $|C_{i,k}| + |C_{k+1,j}|$, $i < k < j$. We assume that the number

Input: *policy P with TCAM space M of depth d, TCAM space A* ($A > M$)
Output: *policy* P' *equivalent to P with TCAM space of at most A*

Perform *binary search* on the policy depth between 1 and d by applying either *SOAW* or *SOAG* on P depending on the structure of P's hierarchy and find the minimum depth l for which the TCAM space of the produced policy P' does not exceed A. Output P'.

Fig. 12. Speed optimization algorithm.

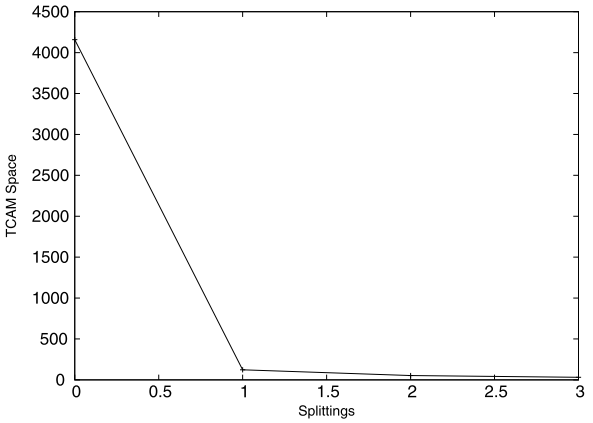

Fig. 13. Required TCAM space as a function of a number of splittings.

of barriers b in any class chain $\mathcal{C}$ is less than $|\mathcal{C}|$; otherwise, no feasible solution exists. The generalized version of *LSA* algorithm for the case of well-structured chain will find optimal placement of $l - b - 1$ splittings. Moreover, the modified version of *LSA* will not try to place splittings at the predefined barrier positions. The generalized version of *LMA* algorithm for the case of general hierarchical chains can just skip during dynamical programming levels corresponding to barriers.

8. Speed optimization

In this section we study the problem of minimizing the number of levels in the policy hierarchy subject to the constraint on the maximum TCAM space. We utilize the space optimization algorithms from the previous section. The speed optimization algorithm is presented on Fig. 12.

Generally speaking, we need to find the optimal value l of the policy depth and then optimize the TCAM space of the transformed policy P' for depth l. The binary search is performed because this value is not known in advance. Once we have found the optimal depth, the space optimization algorithm guarantees minimization of the TCAM space. The running time of the speed optimization algorithm is $O(\log d)$ times the running time of *SOAW* or *SOAG*, that is at most $O(d^3)$ times the number of terminal classes in P, respectively. Remarkably, the running time of the speed optimization algorithm does not depend on A, which can be by orders of magnitude larger than d.

For the case of arbitrary structures the generalized version of *LMA* will not try to place mergings at the predefined barriers positions. We obtain that the time and space complexities of generalized versions of *LSA* and *LMA* are the same as original.

9. Experimental results

In this section, we evaluate the efficiency of the proposed algorithms, which use additional TCAM lookups to minimize the TCAM space required to represent a hierarchical policy. Since hierarchical chains are independent, we evaluate the algorithms for merging/splitting of a single hierarchical chain rather than the whole classification policy. Unfortunately, there is no available for evaluation real-world data for classifiers with more than two levels of hierarchy. Hence, we synthesize data for our experimental study.

In our simulations we generate hierarchical chains with a length of 4. The cardinality of any class belonging to such a chain is at most 16 being chosen uniformly at random. In addition, for the case of general hierarchical chains that are used for simulation of *LMA* algorithm, the intersection between any two neighboring classes is also chosen uniformly at random. For each simulated algorithm we perform over 1000 trials.

Firstly, we consider *LSA* algorithm. Fig. 13 presents the dependency of TCAM space on the number of splittings for the case of well-structured class chains. The simulation results demonstrate very fast exponential growth of the required TCAM space when the number of splittings in a hierarchical chain decreases.

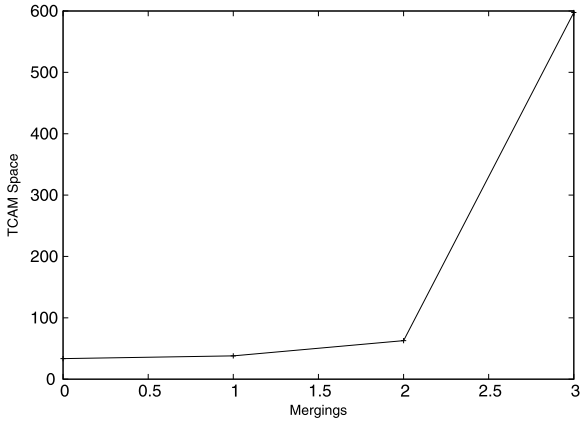

Fig. 14. Required TCAM space as a function of a number of mergings.

Next, we evaluate the *LMA* algorithm for the case of general class chains. Fig. 14 shows the dependency of TCAM space on the number of mergings. As in the previous case, the simulation demonstrates an exponential growth of the required TCAM space with increasing of the number of mergings. Note that as the cardinality of the intersections between classes of the general hierarchical chain is growing, the TCAM space requirement drops down.

As evidenced by the performed experiments for synthetically created classifiers, there exists a fundamental tradeoff between the TCAM classifier space requirements and the lookup speed, which is crucial for efficient implementation of hierarchical classification. Specifically, we need to find the smallest memory footprint that still satisfies the latency requirement so as to optimize the memory cost.

10. Conclusion

Hierarchical packet classification is a key operation needed in provisioning of many crucial network services. One of the major challenges in design of the next generation high-speed switches is to deliver wire-speed packet classification. TCAMs are the dominant industry standard used for multi-gigabit classifiers. However, as packet classification policies grow in depth and complexity, there arises a fundamental tradeoff between the TCAM space and the number of lookups for hierarchical policies.

In this paper we propose novel algorithms based on dynamic programming for solving two important problems concerned with hierarchical packet classification. The algorithms for the first problem minimize the TCAM space given a constraint on the policy depth while the algorithm for the second problem minimizes the policy depth subject to the constraint on the maximum TCAM space. Also we study extensions of space optimization problem. Our algorithms do not require any modification to existing packet classification systems and can be easily deployed. Exploring tradeoff between required TCAM space and performance states a good balance for the future efficient implementation of hierarchical classifications that do not require any hardware changes. As far as we aware, this is the first work to study TCAM speed and space optimization for hierarchical packet classification. We believe that studying of the proposed tradeoffs and interconnection between them is interesting and provides additional insight on generalized classification problem with several hierarchical levels.

References

[1] B. Agrawal, T. Sherwood, Ternary CAM power and delay model: extensions and uses, IEEE Trans. Very Large Scale Integration (VLSI) Systems 16 (5) (May 2008) 554–564.
[2] F. Baboescu, S. Singh, G. Varghese, Packet classification for core routers: Is there an alternative to CAMs?, in: Proc. of IEEE INFOCOM, 2003.
[3] F. Baboescu, G. Varghese, Scalable packet classification, in: Proc. of ACM SIGCOMM, 2001.
[4] J. Bolaria, L. Gwennap, A guide to search engines and networking memory, http://www.linleygroup.com/reports/memory_guide.html.
[5] R. Cohen, D. Raz, Simple efficient TCAM based range classification, in: Proc. of IEEE INFOCOM, 2010.
[6] Sarang Dharmapurikar, Haoyu Song, Jonathan S. Turner, John W. Lockwood, Fast packet classification using bloom filters, in: ANCS 2006, 2006, pp. 61–70.
[7] H.J. Chao, Next generation routers, Proc. IEEE 90 (9) (2002) 1518–1558.
[8] P. Gupta, K. Etzel, J. Bolaria, A scalable and cost-optimized search subsystem for IPv4 and IPv6, http://www.eetimes.com/netseminar.html.
[9] P. Gupta, N. McKeown, Packet classification on multiple fields, in: Proc. of ACM SIGCOMM, 1999, pp. 147–160.
[10] P. Gupta, N. McKeown, Packet classification using hierarchical intelligent cuttings, in: Proc. of Hot Interconnects VII, Aug. 1999.
[11] P. Gupta, N. McKeown, Algorithms for packet classification, IEEE Network (March/April 2001) 24–32.
[12] A. Bremler-Barr, D. Hay, D. Hendler, B. Farber, Layered interval codes for tcam-based classification, ACM SIGMETRICS Performance Evaluation Review 36 (1) (June 2008).
[13] A. Bremler-Barr, D. Hendler, Space-efficient TCAM-based classification using gray coding, in: INFOCOM 2007, 2007, pp. 1388–1396.
[14] K. Lakshminarayanan, A. Rangarajan, S. Venkatachary, Algorithms for advanced packet classification with ternary cams, in: Proc. of ACM SIGCOMM, Aug. 2005, pp. 193–204.

[15] H. Liu, Efficient mapping of range classifier into ternary-cam, in: Proc. of Hot Interconnects, 2002, pp. 95–100.
[16] J. van Lunteren, T. Engbersen, Fast and scalable packet classification, IEEE J. Selected Areas Commun. 21 (4) (2003) 560–571.
[17] Chad R. Meiners, Alex X. Liu, Eric Torng, TCAM Razor: A systematic approach towards minimizing packet classifiers in TCAMs, in: Proc. of the 15th IEEE International Conference on Network Protocols (ICNP), 2007, pp. 226–275.
[18] M.H. Overmars, A.F. van der Stappen, Range searching and point location among fat objects, J. Algorithms 21 (3) (1996) 629–656.
[19] R. Panigrahy, S. Sharma, Reducing TCAM power consumption and increasing throughput, in: Proc. of Hot Interconnects, 2002, pp. 107–112.
[20] S. Singh, F. Baboescu, G. Varghese, J. Wang, Classification using multidimensional cutting, in: Proc. of ACM SIGCOMM, 2003.
[21] V. Srinivasan, G. Varghese, S. Suri, M. Waldvogel, Fast and scalable layer four switching, in: Proc. of ACM SIGCOMM, 1998.
[22] D.E. Taylor, Survey and taxonomy of packet classification techniques, ACM Comput. Surv. (2005) 238–275.
[23] D.E. Taylor, J. Turner, Scalable packet classification using distributed crossproducting of field labels, Technical Report WUCSE-2004-38, Washington Univ., St. Louis, 2004.
[24] G. Varghese, Network Algorithmics: An Interdisciplinary Approach to Designing Fast Networked Devices, Morgan Kaufmann Publishers Inc., 2004.
[25] T.Y.C. Woo, A modular approach to packet classification: Algorithms and results, in: Proc. of IEEE INFOCOM, 2000, pp. 1213–1222.
[26] W. Wu, Jian Shi, Ling Zuo, Bingxin Shi, Power-efficient TCAMS for bursty access patterns, IEEE Micro 25 (4) (July 2005) 64–72.
[27] Cisco ASR 1000 Series Aggregation Services Routers, http://www.cisco.com/en/US/products/ps9343/index.html.
[28] Modular QoS CLI Three-Level Hierarchical Policer, http://www.cisco.com/en/US/docs/ios/12_2t/12_2t13/feature/guide/ft3level.html.
[29] Juniper Networks E320 Broadband Services Router, https://www.apac-juniper.net/juniper_public/campaign/pdf/iptv/en/100103.pdf.
[30] Technical White Paper for MSCG Hierarchical – QoS, www.huawei.com/products/datacomm/pdf/view.do?f=916.
[31] Cypress Semiconductor Corp. Content addressable memory, http://www.cypress.com/.
[32] Integrated Device Technology, Inc. Content addressable memory, http://www.idt.com/.
[33] Netlogic Microsystems. Content addressable memory, http://www.netlogicmicro.com.